Relationships: computational thinking, pedagogy of programming, and Bloom's Taxonomy

Cynthia C. Selby
University of Southampton
Highfield
Southampton UK
44 (0)23 8059 2611
C.Selby@southampton.ac.uk

ABSTRACT

This study explores the relationship between computational thinking, teaching programming, and Bloom's Taxonomy. Data is collected from teachers, academics, and professionals, purposively selected because of their knowledge of the topics of problem solving, computational thinking, or the teaching of programming. This data is analysed following a grounded theory approach. A computational thinking taxonomy is developed. The relationships between cognitive processes, the pedagogy of programming, and the perceived levels of difficulty of computational thinking skills are illustrated by a model.

Specifically, a definition for computational thinking is presented. The skills identified are mapped to Bloom's Taxonomy: Cognitive Domain. This mapping concentrates computational skills at the application, analysis, synthesis, and evaluation levels. Analysis of the data indicates that abstraction of functionality is less difficult than abstraction of data, but both are perceived as difficult. The most difficult computational thinking skill is reported as decomposition. This ordering of difficulty for learners is a reversal of the cognitive complexity predicted by Bloom's model. The plausibility of this inconsistency is explored.

The taxonomy, model, and the other results of this study may be used by educators to focus learning onto the computational thinking skills acquired by the learners, while using programming as a tool. They may also be employed in the design of curriculum subjects, such as ICT, computing, or computer science.

CCS Concepts

• **Social and professional topics~Computational thinking**

Keywords

Computational thinking, pedagogy, programming, Bloom's Taxonomy.

WiPSCE '15, November 09 - 11, 2015, London, United Kingdom

ACM 978-1-4503-3753-3/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2818314.2818315

1. INTRODUCTION

Shortages in science, technology, engineering, and mathematics (STEM) skills are currently widespread in the work force. More than half of employers (58%), responding to the CBI's annual survey, have concerns about their ability to recruit enough highly skilled employees [6]. The Royal Academy of Engineering report, 'ICT for the UK's Future' states "It is essential that a significant proportion of the 14-19 age group understands Computing concepts – programming, design, problem solving, usability, communications and hardware" ([41], p. 17). The Royal Society [42] has indicated that computational thinking, the skills necessary for applying the tools of computer science to understanding the world around us, is actually changing the scientific disciplines themselves and the needs of those engaged in those disciplines. These external pressures on education are not new. Education policy is acknowledged by Dijkstra to be "… hardly influenced by scientific considerations derived from the topics taught, and almost entirely determined by extra-scientific circumstances such as the combined expectations of the students, their parents and their future employers …" ([12], p. 19). These pleas from industry highlight the importance of providing opportunities for learners to acquire knowledge, understanding, and skills associated with programming and problem solving. This setting provides the context for an investigation into the relationship between the teaching of programming and its effect on the acquisition of computational thinking skills by learners.

Specifically, this research attempts to respond to these questions:

- Is there a taxonomy of computational thinking skills?
- What beginning programming skills are most difficult for learners to master?
- What computational thinking skills are most difficult for learners to master?

A grounded theory approach is used to develop a model of the relationships between Bloom's Taxonomy, computational thinking terms, and programming skills. This model is based on the views of 255 participants. The majority described themselves as teachers in areas associated with computing or computer science. They include 39 teaching at the post-graduate level, 43 at the higher education level, 28 at the post-16 level, 126 at a combined secondary and post-16 level, 42 at secondary only, and 19 at primary. An online questionnaire, a community of practice online forum, and interviews were used to collect the data. The model was derived using iterative processes of data collection, qualitative analysis using NVivo data analysis software, and model refinement.

This research contributes to the body of knowledge that may be used to inform the issue of effective teaching strategies for both programming and computational thinking. In the classroom, teachers may employ the results of this study to redesign their own practices to focus on the broader skills of computational thinking, rather than the quite specific skills of mastering a programming language. By identifying a classification of computational thinking skills, curricula could be designed to develop those skills across longer time spans, similar to the teaching of mathematics across twelve years. This paper presents an analysis of this work and a reflection on the model derived from the data analysis.

2. FRAMEWORKS

Two educational taxonomies, which form the backdrop to this research, are Bloom's Taxonomy: Cognitive Domain [4] and the SOLO taxonomy [3]. An additional model specifically addresses the digital domain, Bloom's Digital Taxonomy [9].

These frameworks support this research in varied ways. Cognitive processes can be ordered into taxonomies, indicating increasing complexity [2; 4]. The tasks associated with the digital world can be assigned to these levels [9]. Computational thinking is assumed to be a group of cognitive processes, associated with the digital world [20; 35; 46]. Therefore, it may be possible to order this group of processes into a taxonomy. Successful programming tasks, also associated with the digital world, are assumed to require some cognitive processes [15; 24; 27]. It may be possible to order these separate tasks and cognitive processes into a hierarchy. Even though there is currently no agreed definition of computational thinking [11; 21; 35; 36; 48], these works may afford structures against which possible definitions can be measured. Computational thinking and programming are evidenced by tasks which, to be successful, may need the full range of cognitive processes described in the SOLO taxonomy [7; 8]. This would make SOLO suitable for assessing computational thinking and programming, but not necessarily appropriate for contributing to a search for a definition of computational thinking.

3. LITERATURE REVIEW

Because this research explores the relationships between computational thinking, the teaching of programming, and educational taxonomies, it is appropriate to establish an understanding of each as distinct areas.

3.1 Defining computational thinking

One of the unanswered challenges, presented by Wing [46] in her use of the phrase computational thinking, is the actual definition of the term. From the more recent literature [11; 21; 35; 36; 48] it is evident that there is still confusion over an acceptable definition for the term.

Guzdial [21] has even suggested that a very broad definition of computational thinking is acceptable. Such acceptance shifts the focus away from what computational thinking is to how computational thinking can be taught and how evidence of its acquisition might be observed in learners. A proposed definition of computational thinking, sufficient to support this research, includes decomposition, abstraction, algorithm design, generalisation, and evaluation.

Abstraction is defined as the ability to decide what details of a problem are important and what details can be ignored [47]. In computing, multiple layers of abstraction are often used to reduce the level of complexity of a problem or a representation. Denning [44] acknowledges that abstraction plays an important part in computing, including programming. In programming, an abstraction can be a procedure, function, or data structure. Several participants in the workshop on the scope and nature of computational thinking concur that computational thinking has a focus around the process of abstraction, creating them and defining the relationships between them [35].

Decomposition is defined as breaking down into smaller, more easily solved, parts. It is required when dealing with large problems, complex systems, or complex tasks. Edelson points out that the creation of solutions requires breaking problems down into chunks of particular functionality and sequencing the chunks [36]. Most recently, in refining his own definition of computational thinking, Guzdial [22] includes the use of tools including abstraction and decomposition. Decomposition is not only a suggested component of computational thinking, but also of the classic problem solving techniques espoused by Pólya [37].

Generalisation is a powerful component of problem solving. It describes the ability to express a problem solution in generic terms, which can be applied to different problems that share some of the same characteristics as the original. This definition fits Pólya's description of analogy, the ability to solve a problem based on the known solution to a similar problem [37]. The ability to recognise parts of solutions that have been used in previous situations or that might be used in future situations is included by Kolodner in a definition of computational thinking [36]. These parts, or functional pieces, can be used to solve the current problem or combined in different ways to solve new problems [36].

Being able to identify patterns in both data [19] and across problems [37] is, by some, offered in the definition of computational thinking. Muller [34] found that undergraduates who recognised patterns in problem solutions while programming games were able to recognise and transfer the solution patterns to science simulations. Pattern recognition may be considered a specific type of generalisation.

In her original article, Wing [46] does not use the term algorithmic thinking, preferring the word heuristic instead. However, by 2011, she extends her definition of computational thinking to include algorithmic and parallel thinking [48]. Moursund [35] suggests that computational thinking is related to the idea of procedural thinking, as proposed by Papert in *Mindstorms*. He defines a procedure as a step-by-step set of instructions that can be carried out by a device. The same theme is continued by Sussman [35], who defines computational thinking as a way of devising explicit instructions for accomplishing tasks. Inclusion of algorithmic thinking in a curriculum for high schools appears prior to Wing's contribution. In the Israeli computer science curriculum, Gal-Ezer, et al. [16] placed an emphasis on inclusion of the study of algorithmic processes.

In her initial article, Wing [46] expresses the need for a computational thinker to make trade-offs, by evaluating the use of time and space, power and storage. This evaluation of algorithmic processes, including their power and limitations, is foreshadowed by Gal-Ezer, et al. [16]. Application of the term to user interfaces is evidenced in the second objective of the New Zealand proposed curriculum, as part of designing programs [1]. In their IT approach, L'Heureux, et al. [25] include the ability to evaluate processes, in terms of efficiency and resource utilisation, and the ability to recognise and evaluate outcomes.

Therefore, a definition of computational thinking, adequate to support this research, includes decomposition, abstraction, algorithm design, generalisation, and evaluation.

3.2 Teaching Programming

Learning to program is difficult. There are several reasons identified for this difficulty. These include an inaccurate understanding of how a computational model works; an inability to master reading, tracing, and writing code; and an inability to understand high-level concepts such as design.

There is support for the idea that learners, with an inaccurate understanding of how a computing device actually executes a program, find learning to program particularly difficult [31; 33]. They do not understand and do not create programs that properly handle the fact that any instruction is executed in the state left by the last instruction [13; 26]. Du Boulay [13] suggests that enforcing the idea that there is a strict set of rules governing program execution and avoiding the use of anthropomorphic language should aid in helping learners form an accurate understanding of how the machine works. A significant move from inaccurate to accurate programming concept models was shown by Ma, et al. [31], when using a visualisation tool to introduce cognitive conflict and challenge learners with inappropriate models. This parallels the way in which a one-to-one session with an expert might work, where the expert observes and questions the learner, specifically in the instance when they evidence an inaccurate understanding, to guide their reasoning down a more accurate path. They reported that about half the students with non-viable models moved to a more viable model after using the visualisation tool along with the cognitive conflict technique [31]. Without doubt, an inaccurate understanding of how a computer executes a program will lead the beginners to great difficulties in learning to program.

Lister has been involved in trying to explain the relationship between reading, tracing, explaining, and writing code for many years, most recently in the research of Lopez, et al. [30], and Lister, Fidge, and Teague [28]. When investigating a multilevel hierarchy of programming, based on an analysis of exam papers, strong evidence revealed the association between tracing and writing, especially within the concepts of loops [30]. They also found that hierarchically, data and basics were the foundation, which influenced simple tracing and the understanding of sequences. Mastering these concepts influenced the ability to explain and the ability to write code [30]. This work was built upon in a further study in which Lister, Fidge, and Teague [28] found that effective programmers had developed good tracing skills prior to good writing skills, and that good students can explain the purpose of code without stating what it does line by line. This led them to conclude that writing good effective code requires both tracing and explaining skills [28].

The most difficult concepts to understand are high-level, involving larger entities as opposed to individual details. Perhaps this is because students find it difficult to move away from a line-by-line interpretation of the programming process [26]. Logical thinking is included as a high-level concept by [5]. They have pointed out the connection between the difficulty of topics and the amount of feedback they receive. Design is the most difficult for students and receives the least feedback; syntax is not so difficult but receives large amounts of feedback [5]. They suggest that the emphasis needs to be reversed if students are to master more high-level concepts. In the opinion of Jenkins [23], students demonstrate an inability to cope with multiple problem-solving issues at once and the precision necessary to instruct the computer to carry out the problem-solving algorithm. This inability is exemplified by learners who can read and interpret code, but cannot write their own [23]. Sakhnini and Hazzan [39] conducted one of the few research efforts with high school students using high-level problem-solving concepts. They suggest that the students rely heavily on analogy and should be challenged with false analogies, that students should be taught abstract data type behaviours before implementing them, and that students should be exposed to many problems that can be solved using different strategies. Sakhnini and Hazzan [39] tie their study of abstraction directly to general problem-solving skills and strategies such as those advocated by Pólya [37].

An inaccurate understanding of how a computational model works; an inability to master reading, tracing, and writing code; and an inability to understand high-level concepts such as design are among the reasons identified as contributing to the difficulty of learning to program.

3.3 Educational taxonomies in programming

The appropriateness of using taxonomies in research into programming is supported in several studies. Two of these taxonomies are Bloom's Taxonomy and the SOLO taxonomy.

First year undergraduate students' thinking skills were investigated by Fitzgerald, Simon, and Thomas [14]. They employed a multiple-choice question instrument and a think aloud problem-solving instrument in an effort to determine how students read and understand code. Their results indicated that, overall, students did use strategies, but that no single one was dominant, that students used multiple strategies for each problem, that students used the same strategy in different ways thereby eliciting different results, and that students used good strategies in poor ways [14]. They mapped the students' strategies to the different levels of the cognitive domain defined in Bloom's Taxonomy. As might be expected, the strategies congregated around the comprehension level. However, there were strategies that mapped to all levels. At the highest level, evaluation, were placed those strategies indicating analysis for deeper meaning. This foreshadows the work of Lister, Fidge, and Teague [28], which identified the explaining of code's problem-solving purpose as different from understanding at a line-by-line level.

The applicability of using the SOLO taxonomy to assess the way in which programming students read code was tested by Lister, et al. [29] in their study of undergraduate students. They tested novice programmers only, using a 'think out loud' technique. They were able to place students' responses on an appropriate level of the SOLO taxonomy. They concluded that while teachers often focus on aspects of programming associated with the lower levels of the SOLO taxonomy, they should also offer opportunities for eliciting responses at the relational or higher levels. They suggest that this type of response is manifested by "… an ability to read several lines of code and integrate them into a coherent structure …" ([29], p. 122).

These taxonomies were combined, by Meerbaum-Salant, Armoni, and Ben-Ari [32], in their investigation in middle schools using Scratch to teach computer science concepts. The combination addressed their need to acknowledge that understanding the concept of concurrency is more cognitively complex than creating a script to move a sprite.

SOLO is a competency model illustrating the extent to which understanding is mastered. Bloom's is concerned with identifying the highest level of cognitive demand required by a task or question. The combination [32] acknowledges the complexity associated with acquiring and demonstrating understanding. Using the example above, a learner is asked to 'explain concurrency'. The task, 'explain,' is designed to illicit a response at the comprehension level of Bloom's Taxonomy. The complexity of the processes used to form this understanding, which may have

incorporated application or analysis are not acknowledged in the question. The same learner is asked to create a Scratch script to move a sprite around the screen and is given no other scaffolding materials. In order to achieve synthesis, the learner must know how a puzzle works, must comprehend the meaning of the puzzle pieces, must apply the rules enforced by the puzzle joins, must analyse relationships between the script and what is appearing on the screen, and must create the algorithm (not the script) which makes the sprite move.

This research is not focused on how well a learner has mastered a skill, but on identifying the highest level of cognitive demand required by that skill. Therefore, Bloom's Taxonomy has been selected as the basis for an ordering of the skills of programming.

4. DATA COLLECTION

Participants were selected for their ability to give depth to the dataset. They were perceived to have an interest in the research topic. Three different instruments were employed to provide bring both breadth and depth to the dataset. The complete dataset represents the views of 255 participants.

4.1 Sample

The participants were selected because they possess some interest in the teaching of programming, computational thinking, problem solving, or any combination of the three. Broad categories of participants include those teaching programming; those employed in industries where computational thinking skills and programming skills are used; and members of professional communities of practice, representing industry, academia, or education.

This type of selection is biased toward selection of participants who meet some criteria. Cohen, Manion, and Morrison succinctly reason that, "There is little benefit in seeking a random sample when most of the random sample may be largely ignorant of particular issues and unable to comment on matters of interest to the researcher, in which case a purposive sample is vital" ([10], p. 115). In the case of this research, the sample was selected purposively to consist of those who are perceived to have some knowledge and interest in the teaching of computational thinking or programming. As Strauss and Corbin affirm, theoretical sampling is a foundation stone of grounded theory, the approach employed in this research, that, "… enables the researcher to choose those avenues of sampling that can bring about the greatest theoretical return" ([40], p. 202).

The views of 255 respondents are represented in the data. The community of practice yielded 111 respondents; interviews were conducted with 10 participants; questionnaires were completed by 34 participants. Of the total, 201 described themselves as teachers, 88 as academics, 92 as working in industry, 7 as awarding organisation representatives, and 9 as members of professional bodies. Some respondents have more than one role. Of those indicating a teaching role, 39 teach at post-graduate, 43 teach at higher education, 28 teach at post-16, 126 teach at secondary and post-16, 42 teach at secondary, and 19 teach at primary.

4.2 Instruments

Three different data collection instruments were used. There were an online questionnaire, a community of practice discussion forum, and face-to-face interviews. The data from each were added to a single dataset for analysis.

In the case of the first instrument, an online questionnaire, the targeted sample consisted of members of organisations, both national and local to the researcher, whose ideologies promote the teaching of programming or computational thinking skills.

In addition to the online questionnaire, there was an opportunity to include conversational threads from a community of practice online forum. Not every thread was applicable to the research questions. However, the forum was monitored methodically, for applicable threads, during the same time that the online questionnaire remained open. Once purposively chosen for their applicability to the research questions, the entire contents of these threads contributed to the dataset.

From the questionnaire responses and the community of practice conversations, a further purposive selection was made to identify targets for face-to-face interviews. This selection was made on the perceived ability of the respondents to provide in-depth knowledge about the original research questions.

4.3 Dataset

The data was collected over a period of 16 months during 2012 and 2013. The dataset consists of 123,590 total words, made up of 7,464 from questionnaires, 32,410 from interviews, and 83,706 from the community of practice threads. The data was stored and organised using the NVivo qualitative data analysis tool.

5. Method

During the literature review, no model of the relationships between computational thinking, the skills of programming, and educational taxonomies was revealed. In seeking this model, it was appropriate to consider an inductive approach. Grounded theory suited the objective of the research, to discern a model from current practice.

5.1 Grounded theory

This study uses a grounded theory approach employing qualitative data collection methods and qualitative data analysis techniques. The original grounded theory, in the words of Glaser ([17], p. 8), "…is just a simple, straight forward procedural method to induct theory from any type of data…". While observations and interviews may well support grounded theory, Glaser [17] goes on to include other data sources such as conversations, newspapers, books, videos, etc.

Grounded theory stipulates that categories and concepts do not have to be identified before data collection commences, but are allowed to emerge on the way [10]. This approach provides a mechanism for incorporating new ideas from participant responses without being constrained by predefinition. It allows these new ideas and concepts to be explored, even when they fall outside expected responses.

Strauss and Corbin [40], while not departing from the philosophy of the original grounded theory, focused their attention on the use of structured processes and techniques for promoting the emergence of theory. This approach, although criticised for being prescriptive in some aspects, provides the flexibility to deviate from that prescription. By encouraging the mixing of methods and techniques, their grounded theory approach supports researcher creativity and freedom. Their focus on the data and procedures for encouraging the identification of concepts and promoting the emergence of a core variable provides support and assurance. Their allowance for external influences such as researcher knowledge and literature provide a flexible framework in which this study is set.

5.2 Reliability and validity

In dealing with the question of validity in qualitative research, the overriding idea is to determine if it measures what it is purported to measure [38]. In other words, do the instruments and the research as a whole appear to measure what they claim to measure?

Usher, Bryant, and Johnston [45] report that there are three aspects of validity, pre-validation, internal validation, and post-validation. The one most applicable to grounded theory is internal validation. "…this refers to the actual conduct of the research itself as following the precepts of appropriate practices with respect to devising indicators, data collection and analysis" ([45], p. 215). By following the formal rules of enquiry, the research becomes self-validating.

The rules of grounded theory [40] indicate that the data be collected simultaneously with analysis, that constant comparisons are made to previous data, that the theory change as the data dictates, and that the theory is allowed to develop, unforced. In this study, the analysis was performed simultaneously with data collection and a model was developed and amended as data indicated.

In addition, once gaps had been identified in the data, participants with knowledge to fill those gaps were selected. Purposive sampling could also be used to seek out participants with dissenting views. Overall, the ability to be assured of the quality of responses may ensure the validity of the results.

Another consideration in reflecting on reliability and validity of this research is researcher bias. While it may seem improbable that a researcher can enter into data collection without introducing bias, Glaser affirms that with constant comparison, multiple collections, and continuous conceptualisation, any bias is corrected and therefore, the data may be used objectively [18].

In summary, the reliability and validity of this grounded theory study rely on the interpretation of these attributes in terms of qualitative research and grounded theory. Cohen, Manion, and Morrison describe reliability in qualitative research as including "… fidelity to real life, context- and situation-specificity, authenticity, comprehensiveness, detail, honesty, depth of response and meaningfulness to the respondents." ([10], p. 149). In response, assurances are provided that the study is true to life, is context specific, is authentic, is as comprehensive as time allows, is detailed and honest, and the results may directly influence the classroom practices of the respondents.

6. RESULTS

Data analysis afforded the development of three model components: order of teaching programming, mapping of programming items to Bloom's Taxonomy, and a hierarchy of perceived difficulty of computational thinking skills. The final model is a combination of these components.

6.1 Order of teaching programming

Over a period, it was possible to identify a common sequence to the teaching of programming, by analysing the use of 'before', 'after', and ordering in texts relating to teaching. An example response includes "… with most groups at least, they say we need this, we're going to need that … these ingredients. They won't necessarily get them in the right order to start with." This implies decomposition before algorithm design. Of course, the sequence identified here is not the only teaching sequence, but does represent the teaching practice of many participants. This ordering does not imply context, such as programming language or environment. The order of teaching programming is identified as:

1. constructs, facts, types
2. how individual constructs work
3. use programming constructs in contrived contexts
4. discriminate, decompose, abstract
5. create programs, algorithm design
6. test, evaluate

6.2 Teaching programming and Bloom's

The dataset was interrogated to identify where the participants placed the order of teaching programming items in relation to the levels of Bloom's Taxonomy Cognitive Domain. Responses were grouped into concepts and coded to nodes indicating relationships of order. The concepts were expressed in node names using a noun-adverb-noun tuple. The noun-noun pairs map to the familiar terms in Bloom's levels. The adverb places the concept at a level relative to the other nodes, thus illuminating a form of hierarchy. For example, 'analysis is lower than design' and 'decomposition before evaluation' is interpreted to imply an ordering. The order resulting from the distribution of these tuples is shown here.

Table 1. Bloom's levels of teaching programming

Bloom's Taxonomy	Teaching Programming
Evaluation	evaluate, test
Synthesis	create programs, algorithm design
Analysis	abstract, decompose, discriminate
Application	
Comprehension	structures, constructs, facts, types
Knowledge	

6.3 Computational thinking skills: perceived difficulty

In a similar vein, the dataset was analysed to understand which computational thinking skills were perceived to be the most difficult to master. The key relationships identified here included references with comparatives such as 'is harder than', 'is more difficult than', 'is easier than', 'for more able', and 'distinction'. Arranging the relationships results in the following order of perceived difficulty, with 1 being the easiest computational skill to master and 6 being the most difficult.

1. evaluation
2. algorithm design
3. generalisation
4. abstraction of functionality
5. abstraction of data
6. decomposition

6.4 Relationship model

The three sets of relationships, described above, are brought together into a single model representing the relationships between Bloom's Taxonomy Cognitive Domain, computational thinking skills, and the teaching of programming. An unsurprising result, as discussed above, is that the order in which programming skills are taught directly reflects the order of the levels in Bloom's Cognitive Domain. However, the perceived levels of difficulty of the computational thinking skills when mapped to Bloom's Cognitive Domain are a reversal of the expected order.

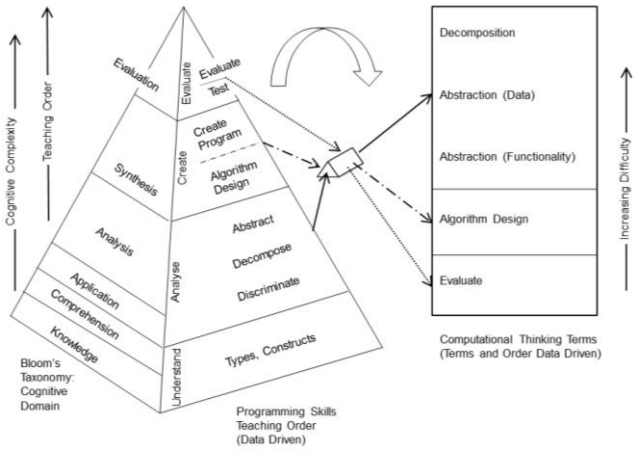

Figure 1. Model: computational thinking, pedagogy of programming, and Bloom's Taxonomy

7. DISCUSSION

This research set out to determine if there was a taxonomy of computational thinking skills, which computational thinking skill is most difficult to master, and which beginning programming skill is most difficult to maser.

Is there a taxonomy of computational thinking skills? A taxonomy of computational thinking skills has been derived from the literature and analysis of the data set. This taxonomy consists of evaluation, algorithm design, abstraction (functionality, data), decomposition, and generalisation (including pattern recognition). It is possible to assign these skills to the levels of Bloom's Taxonomy Cognitive Domain. Evaluation is assigned to the evaluation level; algorithm design is assigned to the synthesis level; abstraction and decomposition are assigned to the analysis level; generalisation is assigned to the application level.

Which computational thinking skill is the most difficult to master? The computational thinking skill perceived as most difficult to master is decomposition. Decomposition is also perceived to be the most difficult programming skill to master. On close inspection, the order of difficulty in mastering the computational thinking skills is a reversal of those same skills mapped to Bloom's Taxonomy Cognitive Domain.

Why is decomposition so difficult? Some reasons are suggested by the participants themselves. These include a lack of experience, incomplete understanding of the problem to solve, and the order of teaching programming. Although learners understand the concept of breaking a problem down, perhaps from a mathematical context, teachers indicate that learners struggle with implementing the process of decomposition. Students appear to be able to use the skill of decomposition more successfully in situations where they already know the solution or understand the problem very well. It may well be that any skill introduced first, when learners are still coping with introductory programming constructs, would reflect the same level of difficulty. However, understanding decomposition, based on the computational thinking taxonomy, is a prerequisite for abstraction, algorithm design, and evaluation. As such, it must be mastered, to some extent, before the complexity of the following levels can be accessed.

Where is generalisation in this model? The term 'generalisation', as a computational thinking skill, is used sparingly in the literature and just as sparingly in the dataset. However, the concept of recognising how small pieces of solutions may be reused and reapplied to similar or unique problems is often identified in both [36]. When generalisation is more broadly interpreted as "where have I seen this type of problem before?" then it is found in the dataset. Generalisation of strategies has been identified by some respondents, for example recognising that ordering is important in some solutions. Generalisation of concepts has also been identified. These examples extend to the ability to understand the fundamentals of one programming language being applied to another and to the behaviour of number systems, such as denary and binary. The key concept identified in the data that is associated with generalisation is the application of knowledge from one domain or context in another. From this, it is logical to place generalisation on the same level as application in Bloom's Taxonomy. In support of this, Bloom purports that "The effectiveness of a large part of the school program is therefore dependent upon how well the students carry over into situations applications which the students never faced in the learning process." ([4], p. 122). The latter part of this statement, as a definition of generalisation, is upheld by the views of the respondents.

What are the next steps? This research has identified that respondents perceive decomposition to be the most difficult computational thinking skill for learners to master. Although possible reasons for this status have been proposed, this research has not revealed why this is the case. Although there is other research concerning the applicability of Bloom's Taxonomy to computer science [14; 43], this particular association deserves further study. This research continues the theme by suggesting that the upper levels of Bloom's Taxonomy are applicable to programming. However, the levels of knowledge and comprehension are yet to be explored to ascertain their contribution to computational thinking.

In closing, the results of this study contribute to the broad areas of research incorporating computational thinking and programming, the more specific area of computer science education research, and the area of computer science pedagogy. In the first instance, a taxonomy of computational thinking skills is proposed to aid understanding of the term. In the second instance, applying Bloom's Taxonomy to the context of programming for 14 – 19 year olds, aids efforts to explore using general education theories in the computer science classroom. In the third instance, the proposed relational model, between levels of cognitive complexity, the teaching of programming skills, and the perceived levels of difficulty of computational thinking skills may be used to influence effective classroom practices.

8. REFERENCES

[1] Bell, T., Andreae, P., and Lambert, L., 2010. Computer Science in New Zealand High Schools. In *Proceedings of the Twelfth Australasian Conference on Computing Education* Australian Computer Society, Inc., Brisbane, Australia, 15-22.

[2] Biggs, J., n.d. SOLO Taxonomy.

[3] Biggs, J. and Collis, K., 1982. *Evaluating the Quality of Learning, The SOLO Taxonomy* Academic Press, Sydney.

[4] Bloom, B., 1956. Taxonomy of Educational Objectives: The Classification of Educational Goals, Handbook 1 Cognitive Domain McKay, New York.

[5] Butler, M. and Morgan, M., 2007. Learning challenges faced by novice programming students studying high level and low feedback concepts. In *Proceedings of the ICT:*

Providing choices for learners and learning. Proceedings ascilite (Singapore2007), www.ascilite.org.au, 99-107.

[6] CBI, 2014. Gateway to Growth: CBI/Pearson education and skills survey 2014(2014).

[7] Chan, C.C., Tsui, M.S., Chan, M.Y.C., and Hong, J.H., 2010. Applying the Structure of the Observed Learning Outcomes (SOLO) Taxonomy on Student's Learning Outcomes: An empirical study. *Assessment & Evaluation in Higher Education 27*, 6 (2002/12/01), 511-527. DOI= http://dx.doi.org/10.1080/0260293022000020282.

[8] Chick, H., 1998. Cognition in the Formal Modes: Research Mathematics and the SOLO Taxonomy. *Mathematics Education Research Journal 10*, 24, 4-26.

[9] Churches, A., 2009a. Bloom's Digital Taxonomy (v3.01), 75.

[10] Cohen, L., Manion, L., and Morrison, K., 2007. *Research Methods in Education*. Routledge, Abingdon, England.

[11] Computing at School Working Group, 2012. Computer Science: A curriculum for schools Computing At School.

[12] Dijkstra, E., 1988. On the cruelty of really teaching computing science The University of Texas at Austin.

[13] Du Boulay, B., 1989. Some difficulties of learning to program. In *Studying the novice programmer*, E. SOLOWAY and J.G. SPOHRER Eds. Lawrence Erlbaum, Hillsdale, NJ, 293-299.

[14] Fitzgerald, S., Simon, B., and Thomas, L., 2005. Strategies that students use to trace code: an analysis based in grounded theory. In *Proceedings of the Proceedings of the first international workshop on Computing education research* (Seattle, WA, USA2005), ACM, 1089793, 69-80. DOI= http://dx.doi.org/10.1145/1089786.1089793.

[15] Fuller, U., Johnson, C.G., Ahoniemi, T., Cukierman, D., Hernán-Losada, Jackova, J., Lahtinen, E., Lewis, T.L., Thompson, D.M., Riedesel, C., and Thompson, E., 2007. Developing a computer science-specific learning taxonomy. *SIGCSE Bull. 39*, 4, 152-170. DOI= http://dx.doi.org/10.1145/1345375.1345438.

[16] Gal-Ezer, J., Beeri, C., Harel, D., and Yehudai, A., 1995. A High School Program in Computer Science. *Computer 28*, 10, 73-80. DOI= http://dx.doi.org/10.1109/2.467599.

[17] Glaser, B., 2009. Jargonizing: The use of the grounded theory vocabulary. In *The Grounded Theory Review*, J. HOLTON Ed. Sociology Press, Mill Valley, CA, USA.

[18] Glaser, B.G., 2002. Constructivist Grounded Theory? *Forum: Qualitative Social Research 3 (3)*(September 2002).

[19] Google, 2011. Exploring Computational Thinking.

[20] Guzdial, M., 2008. Education: Paving the way for computational thinking. *Commun. ACM 51*, 8, 25-27. DOI= http://dx.doi.org/10.1145/1378704.1378713.

[21] Guzdial, M., 2011. A Definition of Computational Thinking from Jeannette Wing. In *Computing Education Blog*, Atlanta.

[22] Guzdial, M., 2012. A nice definition of computational thinking, including risks and cyber-security. In *Computing Education Blog*, Atlanta.

[23] Jenkins, T., 2002. On the Difficulty of Learning to Program. In *Proceedings of the 3rd Annual LTSN-ICS Conference* (Loughborough University2002), The Higher Education Academy.

[24] Johnson, C.G. and Fuller, U., 2006. Is Bloom's taxonomy appropriate for computer science? In *Proceedings of the Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006* (Uppsala, Sweden2006), ACM, 1315825, 120-123. DOI= http://dx.doi.org/10.1145/1315803.1315825.

[25] L'heureux, J., Boisvert, D., Cohen, R., and Sanghera, K., 2012. IT problem solving: an implementation of computational thinking in information technology. In *Proceedings of the 13th Annual Conference on Information Technology Education* ACM, Calgary, Alberta, Canada, 183-188. DOI= http://dx.doi.org/10.1145/2380552.2380606.

[26] Lahtinen, E., Ala-Mutka, K., and Järvinen, H.-M., 2005. A study of the difficulties of novice programmers. In *Proceedings of the Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* (Caparica, Portugal2005), ACM, 1067453, 14-18. DOI= http://dx.doi.org/10.1145/1067445.1067453.

[27] Lister, R., 2000. On blooming first year programming, and its blooming assessment. In *Proceedings of the Proceedings of the Australasian conference on Computing education* (Melbourne, Australia2000), ACM, 359393, 158-162. DOI= http://dx.doi.org/10.1145/359369.359393.

[28] Lister, R., Fidge, C., and Teague, D., 2009. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. In *Proceedings of the Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education* (Paris, France2009), ACM, 1562930, 161-165. DOI= http://dx.doi.org/10.1145/1562877.1562930.

[29] Lister, R., Simon, B., Thompson, E., Whalley, J.L., and Prasad, C., 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In *Proceedings of the Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education* (Bologna, Italy2006), ACM, 1140157, 118-122. DOI= http://dx.doi.org/10.1145/1140124.1140157.

[30] Lopez, M., Whalley, J., Robbins, P., and Lister, R., 2008. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the Proceeding of the Fourth international Workshop on Computing Education Research* (Sydney, Australia2008), ACM, 1404531, 101-112. DOI= http://dx.doi.org/10.1145/1404520.1404531.

[31] Ma, L., Ferguson, J., Roper, M., and Wood, M., 2011. Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education 21*, 1, 57 - 80.

[32] Meerbaum-Salant, O., Armoni, M., and Ben-Ari, M., 2010. Learning computer science concepts with scratch. In *Proceedings of the Proceedings of the Sixth international workshop on Computing education research* (Aarhus, Denmark2010), ACM, 1839607, 69-76. DOI= http://dx.doi.org/10.1145/1839594.1839607.

[33] Milne, I. and Rowe, G., 2002. Difficulties in Learning and Teaching Programming - Views of Students and Tutors.

Education and Information Technologies 7, 1, 55-66. DOI= http://dx.doi.org/10.1023/a:1015362608943.

[34] Muller, O., 2005. Pattern oriented instruction and the enhancement of analogical reasoning. In *Proceedings of the Proceedings of the first international workshop on Computing education research* (Seattle, WA, USA2005), ACM, 1089792, 57-67. DOI= http://dx.doi.org/10.1145/1089786.1089792.

[35] National Research Council, 2010. Report of a Workshop on the Scope and Nature of Computational Thinking The National Academies Press.

[36] National Research Council, 2011. Report of a Workshop of Pedagogical Aspects of Computational Thinking The National Academies Press.

[37] Pólya, G., 1985. *How To Solve It, 2nd ed*. Penguin, London.

[38] Prosser, J., 2004. Ensuring Quality in Qualitative Data. In *Research Methods (part 1 and 2)*, J. SWANN Ed. University of Southampton, School of Education, Southampton, England, 6a.1-6a.27.

[39] Sakhnini, V. and Hazzan, O., 2008. Reducing Abstraction in High School Computer Science Education: The Case of Definition, Implementation, and Use of Abstract Data Types. *J. Educ. Resour. Comput. 8*, 2, 1-13. DOI= http://dx.doi.org/http://doi.acm.org/10.1145/1362787.1362789.

[40] Strauss, A. and Corbin, J., 1998. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage Publications Ltd., London.

[41] The Royal Academy of Engineering, 2009. ICT for the UK's Future: the implications of the changing nature of Information and Communications Technology The Royal Academy of Engineering, London.

[42] The Royal Society, 2012. Shut down or restart? The way forward for computing in UK schools, London.

[43] Thompson, E., Luxton-Reilly, A., Whalley, J.L., Hu, M., and Robbins, P., 2008. Bloom's taxonomy for CS assessment. In *Proceedings of the Proceedings of the tenth conference on Australasian computing education - Volume 78* (Wollongong, NSW, Australia2008), Australian Computer Society, Inc., 1379265, 155-161.

[44] Ubiquity, 2007. An Interview with Peter Denning on the great principles of computing. *Ubiquity 2007*, June, 1. DOI= http://dx.doi.org/10.1145/1276162.1276163.

[45] Usher, R., Bryant, I., and Johnston, R., 1997. *Adult education and the postmodern challenge: learning beyond the limits*. Routledge, London.

[46] Wing, J., 2006. Computational thinking. *Commun. ACM 49*, 3, 33-35. DOI= http://dx.doi.org/10.1145/1118178.1118215.

[47] Wing, J., 2008. Computational thinking and thinking about computing. *Philosophical Transactions of The Royal Society A 366*, 3717-3725. DOI= http://dx.doi.org/10.1098/rsta.2008.0118.

[48] Wing, J., 2011. Research Notebook: Computational Thinking - What and Why? In *The Link* Carneige Mellon, Pittsburgh, PA, 6.